

# 网络安全大作业报告

2019 年 6 月 2 日

作者： 王皓、易婧玮

学号： PB16060885、PB16021503

选题： iptables 及 l7filter 的使用

### 摘要

iptables 是一个常用的包过滤防火墙应用。但是，用户的安全设定实际上是由操作系统内部的一个数据包处理模块（netfilter）执行的，我们可以把 iptables 看做一个客户端代理。iptables 的核心为四表五链。四表为 raw 表、mangle 表、net 表、filter 表。五链为 PREROUTING、INPUT、FORWARD、OUTPUT、POSTROUTING。我们可以向 iptables 中加入不同的过滤规则，实现特定功能。在本次实验中我们测试了 iptables 的基本 accept、drop、reject 动作和 nat 过滤功能。

然而 netfilter 不带有应用层过滤的功能。想要实现应用层协议的过滤，我们选用了 l7filter。l7filter 从 2013 年就没有维护，必须选择较老版本的内核才能成功使用 l7filter 提供的过滤协议，在本次实验中我们选用 centos6.9 进行实验。实验需要先对 linux 内核打补丁，增加 l7filter 的功能模块。完成试验后，能针对性的过滤应用层协议为 http、qq、迅雷等常见的应用层数据包。

关键字：iptables、l7filter、包过滤、防火墙、应用层网关

# 目录

<b>第一部分</b>	<b>iptables</b>	<b>4</b>
<b>1</b>	<b>简介</b>	<b>4</b>
1.1	iptables 与 netfilter . . . . .	4
1.2	链 . . . . .	4
1.3	表 . . . . .	4
1.4	处理动作 . . . . .	5
<b>2</b>	<b>实验</b>	<b>5</b>
2.1	基础命令 . . . . .	5
2.1.1	查看规则命令 . . . . .	5
2.1.2	添加规则命令 . . . . .	6
2.1.3	删除规则 . . . . .	6
2.1.4	修改规则 . . . . .	6
2.1.5	保存规则 . . . . .	7
2.2	实验环境 . . . . .	7
2.2.1	配置路由表 . . . . .	7
2.2.2	验证配置正确 . . . . .	8
2.3	实验内容 . . . . .	9
2.3.1	测试基本功能 . . . . .	9
2.3.2	NAT 功能测试 . . . . .	11
<b>第二部分</b>	<b>l7filter</b>	<b>12</b>
<b>3</b>	<b>实验</b>	<b>12</b>
3.1	准备实验环境 . . . . .	12
3.2	实验内容 . . . . .	13
3.2.1	编译内核 . . . . .	13
3.2.2	安装 iptables . . . . .	15
3.2.3	安装协议特征包 . . . . .	16
3.2.4	开始测试 . . . . .	16
<b>第三部分</b>	<b>小结</b>	<b>20</b>

# 第一部分 iptables

## 1 简介

### 1.1 iptables 与 netfilter

iptables 其实不是真正的防火墙，我们可以把它看做一个客户端代理。用户通过 iptables 这个代理，将用户的安全设定执行到 netfilter 中。netfilter 才是真正的防火墙。netfilter 是 linux 操作系统内部的一个数据包处理模块，它具有如下功能：数据包内容修改、数据包过滤的防火墙功能。

### 1.2 链

进出报文从网卡到达用户空间需要经过多道关卡，这些关卡在 ipatables 中被称为链 (chian)，每个链上都放置了一串规则。图 1 简单形象地描述了一个数据包进出主机时会经过的链：

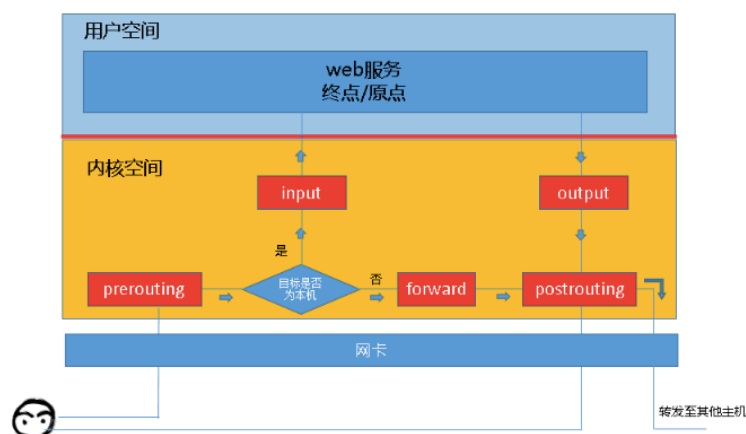


图 1: iptables 链：发送方的数据包到达网卡后，在内核空间内先经过 PREROUTING（转发前）链。当本机的内核支持 IP\_FORWARD 时，数据包的目的地址可能不是本机。若目标为本机，数据包进入 INPUT 链，否则进入 FORWARD（转发）链和 POSTROUTING（转发后）链。本机发送报文时，数据包先经过 OUTPUT 链，再经过 POSTROUTING 链，最终经过网卡转发至其他主机。

### 1.3 表

具有相同功能的规则的集合叫做‘表’。不同功能的规则，我们可以放置在不同的表中进行管理。iptables 已为我们定义了 4 种表：

**filter 表** 负责过滤功能；

**nat 表** 负责网络地址转换功能；

**mangle** 表 拆解报文，作出修改，并重新封装；

**raw** 表 关闭 nat 表上启用的连接追踪机制。

每个链上的规则存在于不同的表如图 2 中：

**PREROUTING** raw 表、mangle 表、nat 表；

**INPUT** mangle 表、filter 表；

**FORWARD** mangle 表、nat 表、filter 表；

**OUTPUT** raw 表、mangle 表、nat 表、filter 表；

**POATROUTING** mangle 表、nat 表。

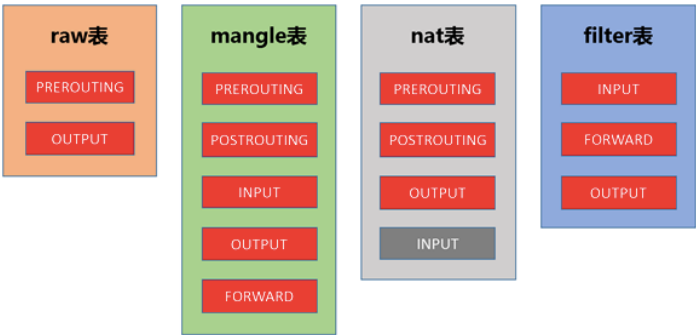


图 2: 不同链上的规则允许存在的表

1.4 处理动作

处理动作在 iptables 被称为 target，常用的有：ACCEPT、DROP、REJECT、SNAT、MASQUERADE、DNAT、REDIRECT、LOG。

2 实验

2.1 基础命令

2.1.1 查看规则命令

查看对应表的所有规则。-v 表示 verbose，详细的。

```
1 iptables -t <表名> (-v) -L
```

查看指定表指定链的所有规则。-n 表示不解析 IP 地址，-line-number 表示显示规则的序号，-x 表示显示计数器的精确值。

```
1 iptables -t (--line-number) <表名> (-nvx) -L <链名>
```

为了方便上述选项常被表示为：

```
1 iptables -line -t <表名> -nvxL (<链名>)
```

### 2.1.2 添加规则命令

在某条链的尾部 (-A) 添加一条规则

```
1 iptables -t <表名> -A <链名> <匹配条件> -j DROP
2 # 将原地址 192.168.1.146 的包丢弃
3 iptables -t filter -A INPUT -s 192.168.1.146 -j DROP
```

在指定链的首部添加一条规则

```
1 iptables -t <表名> -I <链名> <匹配条件> -j <动作>
```

在指定表的指定链的指定位置添加一条规则

```
1 iptables -t <表名> -I <链名> <规则序号> <匹配条件> -j <动作>
```

### 2.1.3 删除规则

按规则序号进行删除

```
1 iptables -t <表名> -D <链名> <规则序号>
```

按照具体的匹配条件与动作进行删除

```
1 iptables -t <表名> -D <链名> <匹配条件> -j <动作>
```

删除指定表中的指定链中的所有规则

```
1 iptables -t <表名> -F <链名>
```

删除指定表的所有规则

```
1 iptables -t <表名> -F
```

### 2.1.4 修改规则

修改指定表中指定链的指定规则

```
1 iptables -t <表名> -R <链名> <规则序号> <原本的匹配规则> -j <动作>
2 iptables -t filter -R 3 -s 192.168.1.146 -j ACCEPT
3 # -s 192.168.1.146 为原本的匹配条件，如果忽略此匹配条件，修改后的规则源地址可能会变成 0.0.0.0/0
```

修改指定表的指定链的默认策略（默认动作），并非修改规则

```
1 iptables -t <表名> -P <链名> <动作>
```

### 2.1.5 保存规则

```
1 # 方法一
2 service iptables save
3 #方法二
4 iptables -save >/etc/sysconfig/iptables
5 iptables -restore </etc/sysconfig/iptables
```

## 2.2 实验环境

使用 vmware fusion 搭建实验环境，使用虚拟主机 <=> 虚拟路由 <=> 虚拟服务器形式。虚拟主机和虚拟路由均为虚拟机，虚拟服务器在本机搭建。

vmware 默认使用 NAT 模式并使用虚拟网卡 vmnet8 与主机共享网络。具体结构如下图所示：

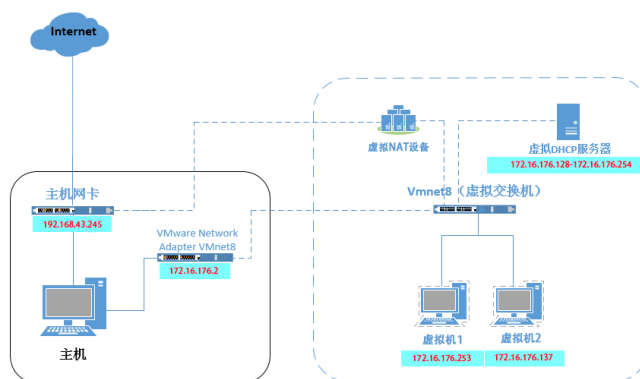


图 3: vmware NAT 模式示意图：我们在 `/Library/Preferences/VMware Fusion/vmnet8/dhcpd.conf` 中配置虚拟 DHCP 服务器分配 ip 地址范围为 172.16.176.128-172.16.176.254。两台试验用虚拟机使用固定 ip 172.16.176.253 和 172.16.176.137。vmnet8 的 ip 为 172.16.176.2。主机的 ip 为 192.168.1.2。

我们使用虚拟机 1 为客户端，虚拟机 2 为路由，主机网卡为服务器端。

### 2.2.1 配置路由表

一开始虚拟机 1 与虚拟机 2 的路由表如下图所示：

```
# yjw @ ubuntu in ~ [4:42:51]
$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.176.2 0.0.0.0 UG 100 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.176.0 * 255.255.255.0 U 100 0 0 ens33
```

(a) 客户端虚拟机路由表

```
temp@ubuntu:~$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.176.2 0.0.0.0 UG 100 0 0 ens33
172.16.176.0 * 255.255.255.0 U 100 0 0 ens33
```

(b) 路由虚拟机路由表

图 4: 虚拟机路由表

配置客户端虚拟机路由表：删除第一条（默认网关为 vmware 虚拟网关），添加默认网关为路由虚拟机。添加方法与结果如下：

```
1 # 删除源默认网关
2 $ sudo route del default
3 # 添加新默认网关
4 $ sudo route add default gw 172.16.176.137
```

最终客户端路由表如下图所示：

```
# yjw @ ubuntu in ~ [5:07:03]
$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.16.176.137 0.0.0.0 UG 0 0 0 ens33
link-local * 255.255.0.0 U 1000 0 0 ens33
172.16.176.0 * 255.255.255.0 U 100 0 0 ens33
```

图 5: 客户端路由表

配置路由虚拟机转发功能：

```
1 temp@ubuntu:~$ sudo sysctl -w net.ipv4.ip_forward=1
```

2.2.2 验证配置正确

我们在客户端虚拟机 ping 服务器主机，使用 traceroute 观察转发路径。测试成功！

```
# yjw @ ubuntu in ~ [5:41:13]
$ traceroute 192.168.1.2
traceroute to 192.168.1.2 (192.168.1.2), 30 hops max, 60 byte packets
1 172.16.176.137 (172.16.176.137) 0.486 ms 0.337 ms 0.547 ms
2 172.16.176.2 (172.16.176.2) 0.453 ms 0.678 ms 0.474 ms
```

图 6: traceroute 测试结果



## 2.3 实验内容

### 2.3.1 测试基本功能

这一阶段我们使用虚拟机 1 和虚拟机 2 进行测试。虚拟机 2 在 8888 和 8889 端口分别跑有两个 http 服务。在虚拟机 1 上 curl 都能得到回显结果：

```
# yjw @ ubuntu in ~ [6:52:08]
$ curl http://172.16.176.137:8888
Hello 8888!
# yjw @ ubuntu in ~ [6:53:05]
$ curl http://172.16.176.137:8889
Hello 8889!
```

图 7: curl 结果

## ACCEPT/DROP icmp 包

设置虚拟机 2 的规则如下

```
1 # 先接受 icmp
2 iptables -A INPUT -p icmp -j ACCEPT
3 iptables -A OUTPUT -p icmp -j ACCEPT
```

此时可以虚拟机 1 可以 ping 通虚拟机 2, 虚拟机 2 也会给虚拟机 1 应答：

No.	Time	Source	Destination	Protocol	Length	Info
8	0.007100	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0x0f15, seq=1/256, ttl=64 (reply in 9)
9	0.008022	172.16.176.137	172.16.176.253	ICMP	98	Echo (ping) reply id=0x0f15, seq=1/256, ttl=64 (request in 8)
14	1.009642	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0x0f15, seq=2/512, ttl=64 (reply in 15)
15	1.010215	172.16.176.137	172.16.176.253	ICMP	98	Echo (ping) reply id=0x0f15, seq=2/512, ttl=64 (request in 14)
20	2.012912	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0x0f15, seq=3/768, ttl=64 (reply in 21)
21	2.013350	172.16.176.137	172.16.176.253	ICMP	98	Echo (ping) reply id=0x0f15, seq=3/768, ttl=64 (request in 20)

(a) wireshark 测试结果

```
# yjw @ ubuntu in ~ [6:58:23] C:/
$ ping 172.16.176.137
PING 172.16.176.137 (172.16.176.137) 56(84) bytes of data.
64 bytes from 172.16.176.137: icmp_seq=1 ttl=64 time=0.628 ms
64 bytes from 172.16.176.137: icmp_seq=2 ttl=64 time=0.735 ms
64 bytes from 172.16.176.137: icmp_seq=3 ttl=64 time=0.711 ms
64 bytes from 172.16.176.137: icmp_seq=4 ttl=64 time=0.604 ms
64 bytes from 172.16.176.137: icmp_seq=5 ttl=64 time=0.721 ms
64 bytes from 172.16.176.137: icmp_seq=6 ttl=64 time=0.862 ms
AC
--- 172.16.176.137 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5011ms
rtt min/avg/max/mdev = 0.604/0.710/0.862/0.084 ms
```

(b) ping 结果

图 8: icmp accept

清空 filter 表规则改为 drop icmp

```

1 # 再 drop icmp
2 iptables -t filter -F
3 iptables -A INPUT -p icmp -j DROP
4 iptables -A OUTPUT -p icmp -j DROP

```

此时虚拟机 1 不能 ping 通虚拟机 2，并且虚拟机 2 无应答：

No.	Time	Source	Destination	Protocol	Length	Info
11	0.373230	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=1/256, ttl=64 (no response...
12	1.394212	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=2/512, ttl=64 (no response...
13	2.417086	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=3/768, ttl=64 (no response...
14	3.441989	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=4/1024, ttl=64 (no respons...
15	4.466989	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=5/1280, ttl=64 (no respons...
18	5.490116	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=6/1536, ttl=64 (no respons...
19	6.514748	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=7/1792, ttl=64 (no respons...
20	7.538501	172.16.176.253	172.16.176.137	ICMP	98	Echo (ping) request id=0xf37, seq=8/2048, ttl=64 (no respons...

(a) wireshark 测试结果

```

# yjw @ ubuntu in ~ [7:05:06] C:1
$ ping 172.16.176.137
PING 172.16.176.137 (172.16.176.137) 56(84) bytes of data.
^C
--- 172.16.176.137 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms

```

(b) ping 结果

图 9: icmp drop: 虚拟机 2 直接丢弃了虚拟机 1 发送的 icmp 包。wireshark 抓包结果分析来看，只有单向发送，没有应答。

## 其他功能测试

过滤敏感词：虚拟机 2 过滤带有 'sex' 的包

```

1 iptables -A INPUT -p tcp --dport 8888 -m string --algo kmp --string "sex" -j DROP

```

```

# yjw @ ubuntu in ~ [7:42:28] C:130
$ curl http://172.16.176.137:8888
Hello 8888!

# yjw @ ubuntu in ~ [7:42:31]
$ curl http://172.16.176.137:8888 -d sex -v
* Rebuilt URL to: http://172.16.176.137:8888/
* Trying 172.16.176.137...
* Connected to 172.16.176.137 (172.16.176.137) port 8888 (#0)
> POST / HTTP/1.1
> Host: 172.16.176.137:8888
> User-Agent: curl/7.47.0
> Accept: */*
> Content-Length: 3
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 3 out of 3 bytes
^C

```

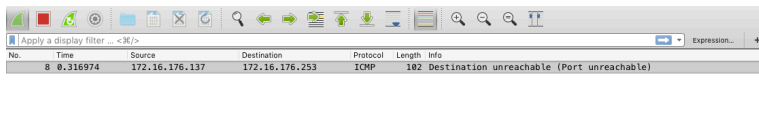
图 10: 过滤敏感词：未添加条件前可以正常访问，添加条件后收不到应答。

虚拟机 reject 8889 端口收到的包：

```

1 iptables -A INPUT -p tcp --dport 8889 -j REJECT

```



No.	Time	Source	Destination	Protocol	Length	Info
8	0.316974	172.16.176.137	172.16.176.253	ICMP	182	Destination unreachable (Port unreachable)

(a) wireshark 结果

```
# yjw @ ubuntu in ~ [7:48:34] C:7
$ curl http://172.16.176.137:8889
Hello 8889!

# yjw @ ubuntu in ~ [7:48:51]
$ curl http://172.16.176.137:8889
curl: (7) Failed to connect to 172.16.176.137 port 8889: Connection refused
```

(b) curl 结果

图 11: reject 8889 测试结果: 虚拟机 1 访问虚拟机 2 的 8889 端口。由 wireshark 测试结果可知虚拟机 2 返回了一个 icmp 包, 告知虚拟机 1 他被 reject。从而 curl 终止访问 (不会和 drop 不停尝试访问)

### 2.3.2 NAT 功能测试

本阶段虚拟机 1 为客户端, 虚拟机 2 为路由, 本机为服务器端。虚拟机 2 不再开启服务, 关闭 8888 和 8889 端口服务。主机在 8889 端口开启 http 服务。

由于虚拟机 2 扮演角色为防火墙, 先置默认规则为 Drop

```
1 sudo iptables -P FORWARD DROP
```

然后设置允许访问外网主机的 8889 端口

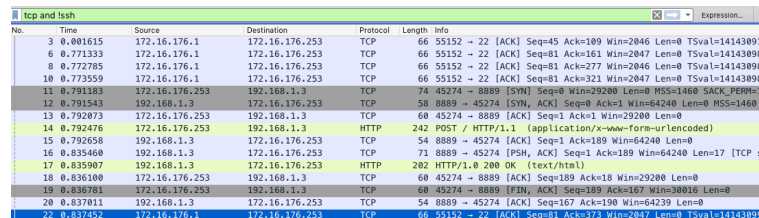
```
1 sudo iptables -I FORWARD -s 192.168.1.2 -p tcp --dport 8889 -j ACCEPT
```

curl 测试能否访问主机 8889 端口:

```
# yjw @ ubuntu in ~ [16:55:17]
$ curl http://192.168.1.3:8889 -d 48927184718246921948621875971205703915
Hello 8889!
```

图 12: curl 测试

wireshark 抓包结果:



No.	Time	Source	Destination	Protocol	Length	Info
3	0.001615	172.16.176.1	172.16.176.253	TCP	66	55152 → 22 [ACK] Seq=45 Ack=109 Win=2046 Len=0 TSval=141430911
6	0.771333	172.16.176.1	172.16.176.253	TCP	66	55152 → 22 [ACK] Seq=81 Ack=161 Win=2047 Len=0 TSval=141430987
8	0.772785	172.16.176.1	172.16.176.253	TCP	66	55152 → 22 [ACK] Seq=81 Ack=277 Win=2046 Len=0 TSval=141430987
10	0.773559	172.16.176.1	172.16.176.253	TCP	66	55152 → 22 [ACK] Seq=81 Ack=321 Win=2047 Len=0 TSval=141430987
11	0.791183	172.16.176.253	192.168.1.3	TCP	74	45274 → 8889 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
12	0.791543	192.168.1.3	172.16.176.253	TCP	58	8889 → 45274 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13	0.792073	172.16.176.253	192.168.1.3	TCP	60	45274 → 8889 [ACK] Seq=1 Ack=1 Win=29200 Len=0
14	0.792476	172.16.176.253	192.168.1.3	HTTP	242	POST / HTTP/1.1 (application/x-www-form-urlencoded)
15	0.792658	192.168.1.3	172.16.176.253	TCP	54	8889 → 45274 [ACK] Seq=1 Ack=189 Win=64240 Len=0
16	0.835468	192.168.1.3	172.16.176.253	TCP	71	8889 → 45274 [PSH, ACK] Seq=1 Ack=189 Win=64240 Len=17 [TCP se
17	0.835907	192.168.1.3	172.16.176.253	HTTP	202	HTTP/1.0 200 OK (text/html)
18	0.836100	172.16.176.253	192.168.1.3	TCP	60	45274 → 8889 [ACK] Seq=189 Ack=18 Win=29200 Len=0
19	0.836981	172.16.176.253	192.168.1.3	TCP	60	45274 → 8889 [FIN, ACK] Seq=189 Ack=167 Win=0 Len=0
20	0.837011	192.168.1.3	172.16.176.253	TCP	54	8889 → 45274 [ACK] Seq=167 Ack=190 Win=64239 Len=0
22	0.837452	172.16.176.1	172.16.176.253	TCP	66	55152 → 22 [ACK] Seq=81 Ack=373 Win=2047 Len=0 TSval=141430994

图 13: wireshark 测试

## 第二部分 17filter

### 3 实验

#### 3.1 准备实验环境

使用 Centos6.9 和 ubuntu16.04 作为测试环境。下载 CentOS-6.9-x86\_64-minimal.iso, Unbuntu16.04.iso。安装 Centos 时出现过以下问题，已解决。

**vmware-tools install failed** 在安装时正会出现 vmware-tools install failed 错误无法进入系统，自定义虚拟机安装。

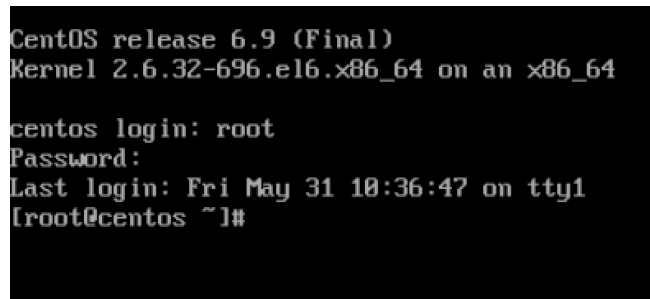


图 14: vmware-tools install failed

**虚拟机未和 mac 处于同一子网内** 原因为 eth0 网卡未设置成 boot 自启,修改/etc/sysconfig/network-scripts/ifcfg-eth0 中的 onboot 改为 yes。重启网络服务后正常 'service network restart', 如下图所示:

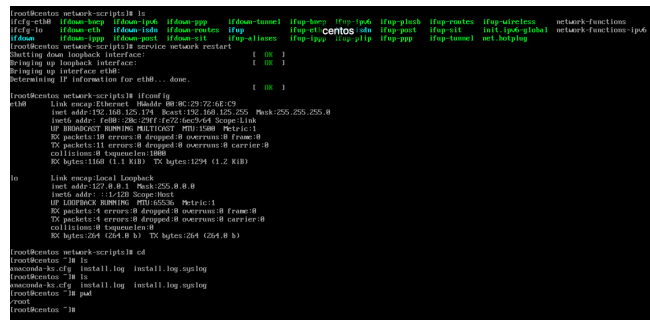


图 15: 重启网络服务

**基本配置** Centos 作为转发端做以下配置：更新源、安装 git、make、gcc、automake、autoconf、libtool 等软件包，配置好 ssh 和 vmware-tools 方便管理使用虚拟机：

1. 修改/etc/yum.repos.d/CentOS-Base.repo 配置文件，使用命令 makecache 更新源；

## 2. 下载软件包安装

```
1 [root@centos ~]# yum install -y zsh git vim gcc automake autoconf libtool make patch && sh -c $(curl -fsSL
https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

3. 修改配置文件 `/etc/ssh/sshd_config` 打开 22 端口的服务, 使用命令重启 `sshd: service sshd restart;`

4. 挂载 `vmware-tools` 光驱在 `/mnt/cdrom` 上, 这是一个只读文件系统, 把压缩包拷出来再解压, 安装好了 `vmware-tools`, 方便管理文件。

## 3.2 实验内容

### 3.2.1 编译内核

准备好 `linux-2.6.35.8.tar.gz` 和 `netfilter-layer7-v2.23.tar.gz` 两个文件来编译内核:

```
1 cd /root tar xf linux-2.6.35.8.tar.gz -C /usr/src \
2 && tar xf netfilter-layer7-v2.23.tar.gz \
3 && ln -s /usr/src/linux-2.6.35.8 /usr/src/linux
4 cd /usr/src/linux patch -p1 < /root/netsec/netfilter-layer7-v2.23/kernel-2.6.35-layer7-2.23.patch
5 cp /boot/config-2.6.32-696.el6.x86_64 /usr/src/linux/.config make menuconfig
```

发现报错: `unable to find the ncurses libraries`。原因是缺少 `ncurses-devel`, 使用 `yum install ncurses-devel` 命令重新安装

编译内核时需要将以下几项编译进入内核模块: `Netfilter connection tracking support`、`"layer7" match support`、`"iprange" address range match support`、`IPv4 connection tracking support (required for NAT)` 和 `Iptables Support`。步骤如下所示:

1. 先进入 `Core Netfilter Configuration: Networking support` ---> `Networking options` ---> `Network packet filtering framework (Netfilter)` ---> `Core Netfilter Configuration` 见到如下界面:

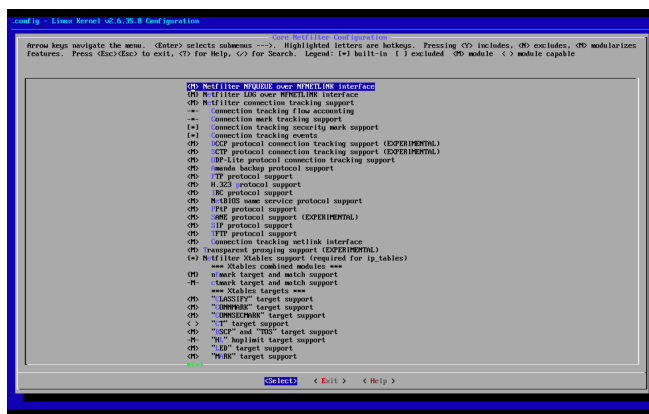


图 16: 内核编译 Core Netfilter Configuration 选项



```

# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE: You have a /boot partition. This means that
# all kernel and initrd paths are relative to /boot/, eg.
# root (hd0,0)
# kernel /vmlinuz-version ro root=/dev/mapper/vg_centos-lv_root
# initrd /initrd-[generic]-lversion.img
#boot=/dev/sda
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.35.8)
    root (hd0,0)
    kernel /vmlinuz-2.6.35.8 ro root=/dev/mapper/vg_centos-lv_root rd_NO_LUK
S LANG=en_US.UTF-8 rd_LUM_LU=vg_centos/lv_swap rd_NO_MD SYSFONT=latarcyrheb-sun1
6 crashkernel=auto rd_LUM_LU=vg_centos/lv_root KEYBOARDTYPE=pc KEYTABLE=us rd_M
0_DM rhgb quiet
    initrd /initramfs-2.6.35.8.img
title CentOS 6 (2.6.32-696.el6.x86_64)
    root (hd0,0)

```

图 18: 编译新内核后的/boot/grub/grub.conf 文件

5. 重新启动后查看新的内核版本，发现已经编译成功了：

```

# root@centos ~# uname -a
Linux centos 2.6.35.8 #1 SMP Fri May 31 13:03:01 HKT 2019 x86_64 x86_64 G
NU/Linux
# root@centos ~#

```

图 19: 重新启动后查看内核版本

### 3.2.2 安装 iptables

1. 下载 iptables-1.4.21.tar.bz2 源码。存入 netsec 目录，开始源码编译安装 iptables：

```

1  mkdir iptables
2  cp /etc/init.d/iptables iptables/ #备份 iptables 启动脚本
3  cp /etc/sysconfig/iptables-config ./ #备份 iptables 配置文档
4  tar xf iptables-1.4.21.tar.bz2 cp netfilter -layer7-v2.23/iptables-1.4.3forward-for-kernel-2.6.20
   forward/libxt_layer7.* iptables-1.4.21/extensions #把 l7filter 的拓展模块移到 iptables 的 extention
   目录
5  cd iptables-1.4.21
6  ./configure --prefix=/usr --with-ksource=/usr/src/linux
7  make # 编译
8  make install # 安装
9  cp iptables-config /etc/sysconfig/iptables-config # 复制配置文档
10 cp iptables/iptables /etc/init.d/iptables # 复制启动脚本

```

- 最后把/etc/init.d/iptables 启动脚本中的如下部分的/sbin 修改为/usr/sbin，如下所示。

```

1 if [ ! -x /usr/sbin/$IPTABLES ]; then
2 echo -n "${IPTABLES}:/usr/sbin/$IPTABLES does not exist."; warning; echo
3 exit 5 fi

```

### 3.2.3 安装协议特征包

- l7filter 过滤各种协议特征需要协议特征包，直接解压安装即可：

```

1 tar xf l7-protocols-2009-05-28
2 cd l7-protocols-2009-05-28
3 make install
4 ls protocols #查看支持的协议，如下所示

```

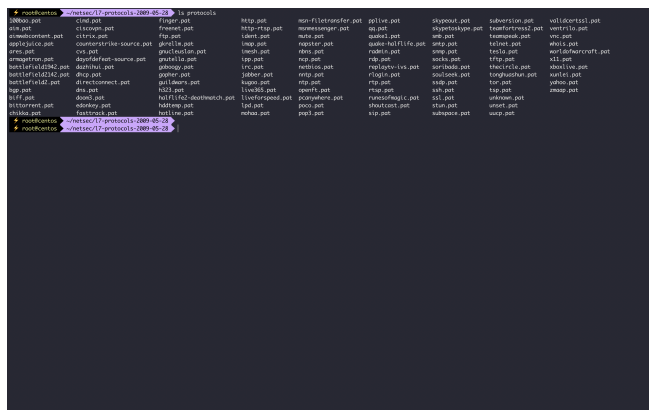


图 20: l7filter 过滤支持协议

### 3.2.4 开始测试

测试环境：ubuntu16.04, centos 6.9. 两台主机在同一子网内，配置 ubuntu 默认网关为 centos，这样 ubuntu 如果需要连接互联网全部都需要通过 centos 转发流量。

- 首先查看 centos 的 ip 地址：192.168.125.174



```

root@centos ~/netsec/L7-protocols-2009-05-28 ifconfig
eth0 Link encap:Ethernet HWaddr 00:0C:29:72:6E:C9
      inet addr:192.168.125.174 Bcast:192.168.125.255 Mask:255.255.255.0
      inet6 addr: fe80::20c:29ff:fe72:6ec9/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:3578 errors:0 dropped:0 overruns:0 frame:0
      TX packets:2161 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:849511 (829.6 KiB)  TX bytes:439248 (428.9 KiB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

```

图 21: ifconfig 查看 centos 的 ip 地址

- 配置 centos 支持转发功能: 修改 /etc/sysctl.conf, 把 net.ipv4.ip\_forward 的值改为 1, 最后使用 sysctl -p 完成修改;

```

1 net.ipv4.ip_forward = 1
2 net.ipv4.conf.default.rp_filter = 1
3 net.ipv4.conf.default.accept_source_route = 0
4 kernel.sysrq = 0
5 kernel.core_uses_pid = 1
6 net.ipv4.tcp_syncookies = 1
7 kernel.msgmnb = 65536
8 kernel.msgmax = 65536
9 kernel.shmall = 4294967296

```

- 配置 ubuntu 设置默认网关为 centos 的 ip 地址, 先查看默认网关可以看到 vmnet8 的 中有一个默认网关 192.168.125.2;

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	192.168.125.2	0.0.0.0	UG	100	0	0	ens33
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	ens33
192.168.125.0	0.0.0.0	255.255.255.0	U	100	0	0	ens33

- 删除这个默认网关后添加 192.168.125.174 为默认网关, 再次查看默认网关;

```

1 sudo route del default gw 192.168.125.2
2 sudo route add default gw 192.168.125.174
3 route -n

```

```
# wanghao @ ubuntu in ~ [8:16:24]
$ route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 192.168.125.2 0.0.0.0 UG 100 0 0 ens33
169.254.0.0 0.0.0.0 255.255.0.0 U 1000 0 0 ens33
192.168.125.0 0.0.0.0 255.255.255.0 U 100 0 0 ens33

# wanghao @ ubuntu in ~ [9:47:43]
$ sudo route add default gw 192.168.125.174
[sudo] password for wanghao:

# wanghao @ ubuntu in ~ [10:16:19]
$ sudo route del default gw 192.168.125.2

# wanghao @ ubuntu in ~ [10:16:25]
$ route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 192.168.125.174 0.0.0.0 UG 0 0 0 ens33
169.254.0.0 0.0.0.0 255.255.0.0 U 1000 0 0 ens33
192.168.125.0 0.0.0.0 255.255.255.0 U 100 0 0 ens33
```

图 22: 修改 ubuntu 的默认网关

## 5. 配置 iptables 进制 http 协议前 ubuntu 的访问情况;

```
$ curl -H 'Cache-Control: no-cache' http://www.ustc.edu.cn
curl: (7) Failed to connect to www.ustc.edu.cn port 80: Connection timed out
```

图 23: 配置丢弃 http 包前的 ubuntu 访问情况

## 6. 然后查看虚拟路由表;

```
1 iptables -A FORWARD -s 192.168.125.0/24 -m layer7 --l7proto http -j DROP
2 iptables -t filter -L
```

```
root@centos ~ # iptables -A FORWARD -s 192.168.125.0/24 -m layer7 --l7proto http -j DROP
root@centos ~ # iptables -t filter -L

Chain INPUT (policy ACCEPT)
target prot opt source destination state
ACCEPT all -- anywhere anywhere state RELATED,ESTABLISHED
ACCEPT icmp -- anywhere anywhere
ACCEPT all -- anywhere anywhere
ACCEPT tcp -- anywhere anywhere state NEW tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target prot opt source destination
DROP all -- 192.168.125.0/24 anywhere LAYER7 l7proto http

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

图 24: 配置丢弃 http 包后的 centos 虚拟路由表

## 7. 配置 iptables 使用 l7filter 进制 http 访问后访问情况如下, 实验成功。

```
$ curl -H 'Cache-Control: no-cache' http://www.ustc.edu.cn
curl: (7) Failed to connect to www.ustc.edu.cn port 80: Connection timed out
```

图 25: 配置丢弃 http 包后的 ubuntu 访问情况

8. l7filter 支持的过滤协议中还包括 qq、迅雷、pop3、telnet、tor、ssl 等。不过因为 l7filter 已经有人在维护了，测试这些协议的时候出现一些 bug。就没有继续深究下去。

## 第三部分 小结

本次试验先是测试了如何使用 iptables。先使用了 iptable 基本的 ACCEPT/DROP/REJECT 功能。通过这些功能，服务器或客户端可以根据规则允许或拒绝访问。然后我们测试了 iptables 的 nat 过滤功能。使用该功能可以对 NAT 网关配置防火墙，限制内网主机访问外网的能力或屏蔽外网发往内网的数据包。在测试 iptables 各项功能的实验中，我们遇到 route 表会被 vmware 强制刷新的问题。为了解决该问题，我们在每次实验前会对 route 表重新检查设定，以保证实验的正常进行。

我们在本次试验中测试了 kernel 版本的 l7filter 的功能，难点主要在编译安装 l7filter。现在应用层防火墙的通用方案很少，因为各种应用层协议太多，通用的解决方案过于庞大难以维护。其次因为要适应于各种协议的应用层防火墙必须从应用层下就开始检测过滤，实际上是需要底层实现的支持的，比如 l7filter 的功能完善的版本就是 kernel 版本而非 userspace 版本。未来要做更好的应用层防火墙需要结合强大的流量分析手段例如使用深度学习去做更加准确的数据包分析才能获得较好的效果。